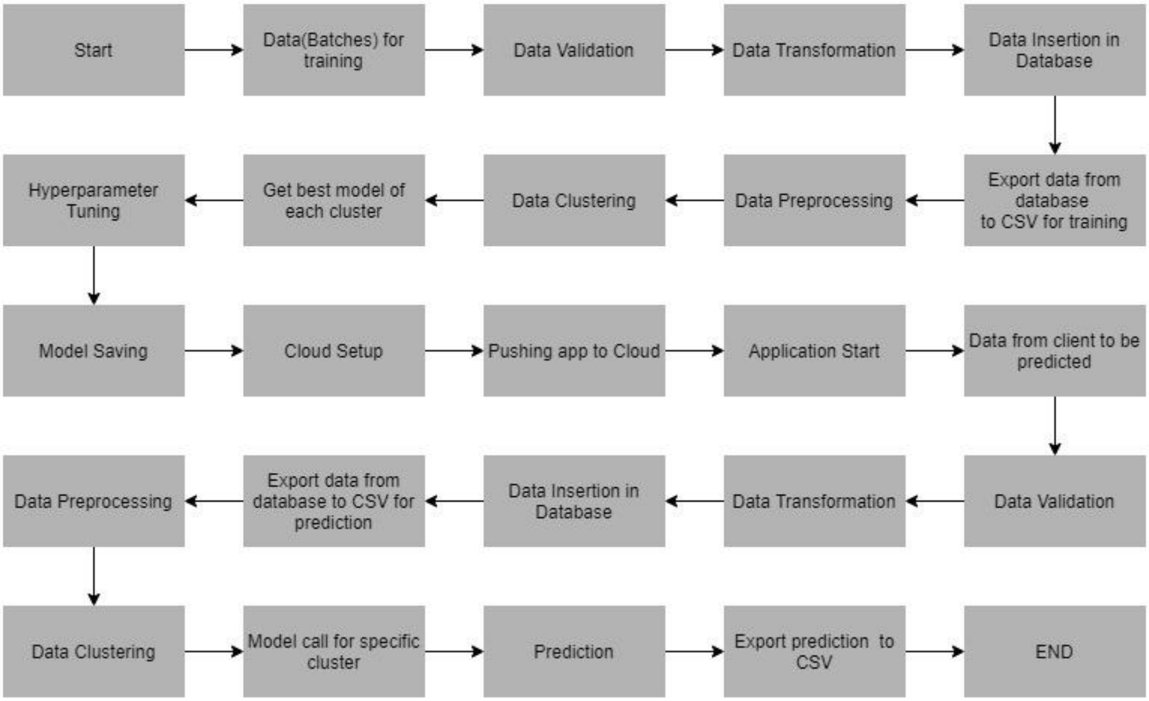
Problem Statement

Build a predictive model and find out the sales of each product at a particular store.

Architecture


Start
Data(Batches) for training
Data Validation
Data Transformation
Data Insertion in Database
Export data from database to CSV for training
Data Preprocessing
Data Clustering
Get best model of each cluster
Hyperparameter Tuning
Model Saving
Cloud Setup
Pushing app to Cloud
Application Start
Data from client to be predicted
Data Validation
Data Transformation
Data Insertion in Database
Export data from database to CSV for prediction
Data Preprocessing
Data Clustering
Model call for specific cluster
Prediction
Export prediction to CSV
END

Data Description

- ıtem_Identifier: Unique product ID
- Item_Weight: Weight of product
- Item_Fat_Content: Whether the product is low fat or not
- Item_Visibility: The % of total display area of all products in a store allocated to the particular product
- Item_Type: The category to which the product belongs
- Item_MRP: Maximum Retail Price (list price) of the product

- Outlet_Identifier: Unique store ID
- Outlet_Establishment_Year: The year in which store was established
- Outlet_Size: The size of the store in terms of ground area covered
- Outlet_Location_Type: The type of city in which the store is located
- Outlet_Type: Whether the outlet is just a grocery store or some sort of supermarket
- Item_Outlet_Sales: Sales of the product in the particulat store. This is the outcome variable to be predicted.

Apart from training files, we also require a "schema" file from the client, which contains all the relevant information about the training files such as:

Name of the files, Length of Date value in FileName, Length of Time value in FileName, Number of Columns, Name of the Columns, and their datatype.

Data Validation

In this step, we perform different sets of validation on the given set of training files.

1. Name Validation- We validate the name of the files based on the given name in the schema file. We have created a regex pattern as per the name given in the schema file to use for validation. After validating the pattern in the name, we check for the length of date in the file name as well as the length of time in the file name. If all the values are as per requirement, we move such files to "Good_Data_Folder" else we move such files to "Bad_Data_Folder."

2. Number of Columns - We validate the number of columns present in the files, and if it doesn't match with the value given in the schema file, then the file is moved to "Bad_Data_Folder."

3. Name of Columns - The name of the columns is validated and should be the same as given in the schema file. If not, then the file is moved to "Bad_Data_Folder".

4. The datatype of columns - The datatype of columns is given in the schema file. This is validated when we insert the files into Database. If the datatype is wrong, then the file is moved to "Bad_Data_Folder".

5. Null values in columns - If any of the columns in a file have all the values as NULL or missing, we discard such a file and move it to "Bad_Data_Folder".

Data Insertion in Database

1) Database Creation and connection - Create a database with the given name passed. If the database is already created, open the connection to the database.

2) Table creation in the database - Table with name - "Good_Data", is created in the database for inserting the files in the "Good_Data_Folder" based on given column names and datatype in the schema file. If the table is already present, then the new table is not created and new files are inserted in the already present table as we want training to be done on new as well as old training files.

3) Insertion of files in the table - All the files in the "Good_Data_Folder" are inserted in the above-created table. If any file has invalid data type in any of the columns, the file is not loaded in the table and is moved to "Bad_Data_Folder".

Model Training

1) Data Export from Db - The data in a stored database is exported as a CSV file to be used for model training.

2) Data Preprocessing

a) Check for null values in the columns. If present, impute the null values using the KNN imputer

b) transform the features using log transformation

c) Scale the training and test data separately

3) Clustering - KMeans algorithm is used to create clusters in the preprocessed data. The optimum number of clusters is selected by plotting the elbow plot, and for the dynamic selection of the number of clusters, we are using "KneeLocator" function. The idea behind clustering is to implement different algorithms

To train data in different clusters. The Kmeans model is trained over preprocessed data and the model is saved for further use in prediction.

4) Model Selection - After clusters are created, we find the best model for each cluster. We are using two algorithms, "Random forest Regressor" and “Linear Regression”. For each cluster, both the algorithms are passed with the best parameters derived from GridSearch. We calculate the Rsquared scores for both models and select the model with the best score. Similarly, the model is selected for each cluster. All the models for every cluster are saved for use in prediction.

Prediction Data Description

Client will send the data in multiple set of files in batches at a given location. Data will contain climate indicators in 8 columns.

Apart from prediction files, we also require a "schema" file from client which contains all the relevant information about the training files such as:

Name of the files, Length of Date value in FileName, Length of Time value in FileName, Number of Columns, Name of the Columns and their datatype.

Data Validation

In this step, we perform different sets of validation on the given set of training files.

1) Name Validation- We validate the name of the files on the basis of given Name in the schema file. We have created a regex pattern as per the name given in schema file, to use for validation. After validating the pattern in the name, we

check for length of date in the file name as well as length of time in the file name. If all the values are as per requirement, we move such files to "Good_Data_Folder" else we move such files to "Bad_Data_Folder".

2) Number of Columns - We validate the number of columns present in the files, if it doesn't match with the value given in the schema file then the file is moved to "Bad_Data_Folder".

3) Name of Columns - The name of the columns is validated and should be same as given in the schema file. If not, then the file is moved to "Bad_Data_Folder".

4) Datatype of columns - The datatype of columns is given in the schema file. This is validated when we insert the files into Database. If dataype is wrong then the file is moved to "Bad_Data_Folder".

5) Null values in columns - If any of the columns in a file has all the values as NULL or missing, we discard such file and move it to "Bad_Data_Folder".

Data Insertion in Database

1) Database Creation and connection - Create database with the given name passed. If the database is already created, open the connection to the database.

2) Table creation in the database - Table with name - "Good_Data", is created in the database for inserting the files in the "Good_Data_Folder" on the basis of given column names and datatype in the schema file. If table is already present then new table is not created, and new files are inserted the already present table as we want training to be done on new as well old training files.

3) Insertion of files in the table - All the files in the "Good_Data_Folder" are inserted in the above-created table. If any file has invalid data type in any of the columns, the file is not loaded in the table and is moved to "Bad_Data_Folder".

Prediction

1) Data Export from Db - The data in the stored database is exported as a CSV file to be used for prediction.

2) Data Preprocessing

a) Check for null values in the columns. If present, impute the null values using the KNN imputer

b) transform the features using log transformation

c) Scale the training and test data separately

3) Clustering - KMeans model created during training is loaded, and clusters for the preprocessed prediction data is predicted.

4) Prediction - Based on the cluster number, the respective model is loaded and is used to predict the data for that cluster.

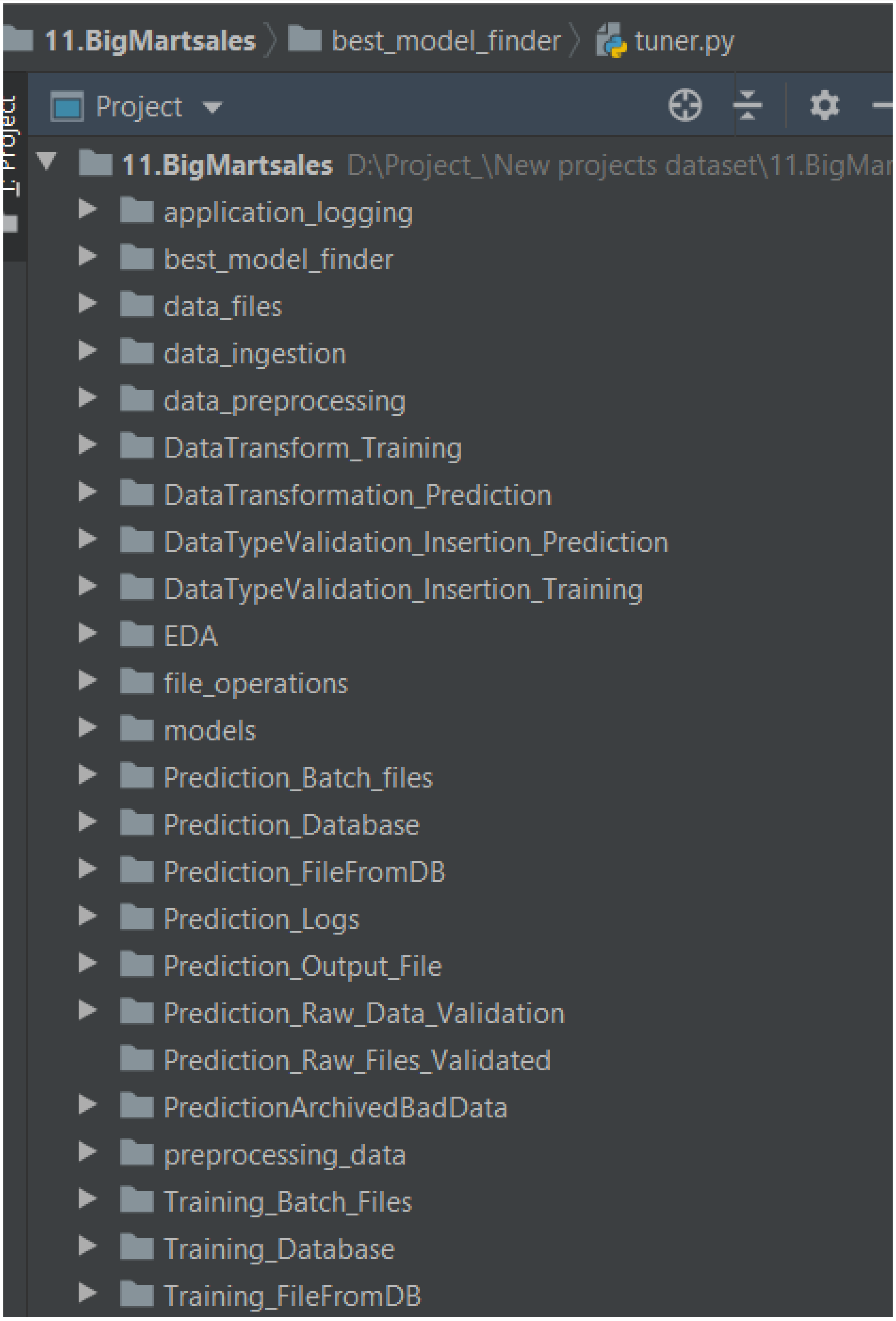
5) Once the prediction is made for all the clusters, the predictions along with the original names before label encoder are saved in a CSV file at a given location and the location is returned to the client.

Deployment

We will be deploying the model to the Pivotal Web Services Platform.

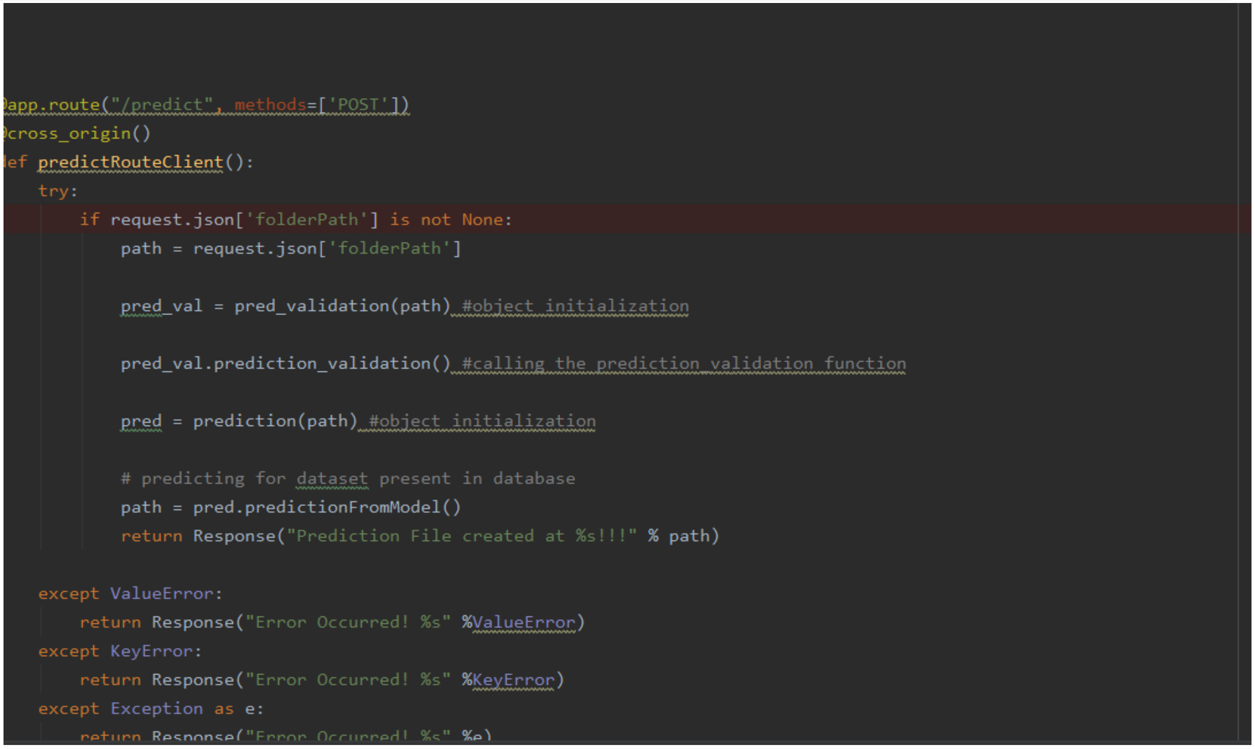
This is a workflow diagram for the prediction of using the trained model.

Now let's see project folder structure.



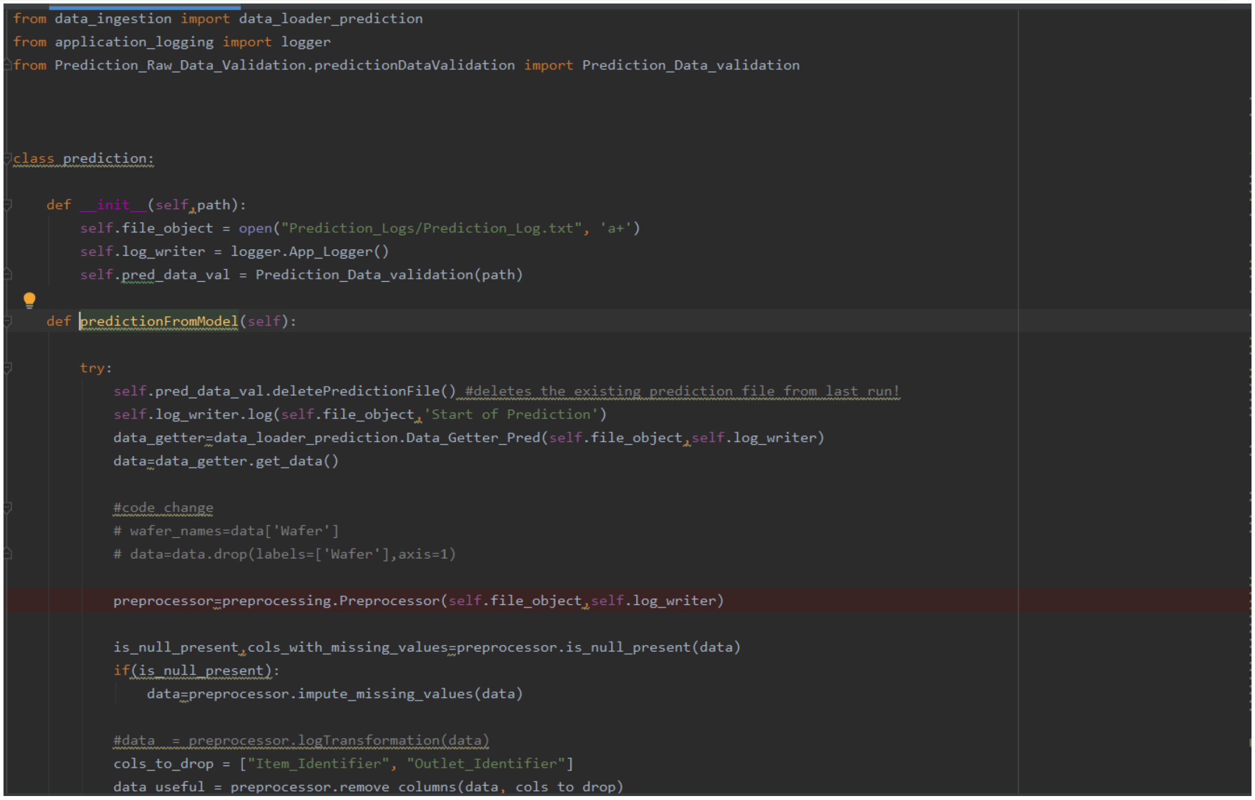
```
▼ 11.BigMartsales D:\Project_\New projects dataset\11.BigMar
    ▶ application_logging
    ▶ best_model_finder
    ▶ data_files
    ▶ data_ingestion
    ▶ data_preprocessing
    ▶ DataTransform_Training
    ▶ DataTransformation_Prediction
    ▶ DataTypeValidation_Insertion_Prediction
    ▶ DataTypeValidation_Insertion_Training
    ▶ EDA
    ▶ file_operations
    ▶ models
    ▶ Prediction_Batch_files
    ▶ Prediction_Database
    ▶ Prediction_FileFromDB
    ▶ Prediction_Logs
    ▶ Prediction_Output_File
    ▶ Prediction_Raw_Data_Validation
      Prediction_Raw_Files_Validated
    ▶ PredictionArchivedBadData
    ▶ preprocessing_data
    ▶ Training_Batch_Files
    ▶ Training_Database
    ▶ Training_FileFromDB
```

requirements.txt file consists of all the packages that you need to deploy the app in the cloud.



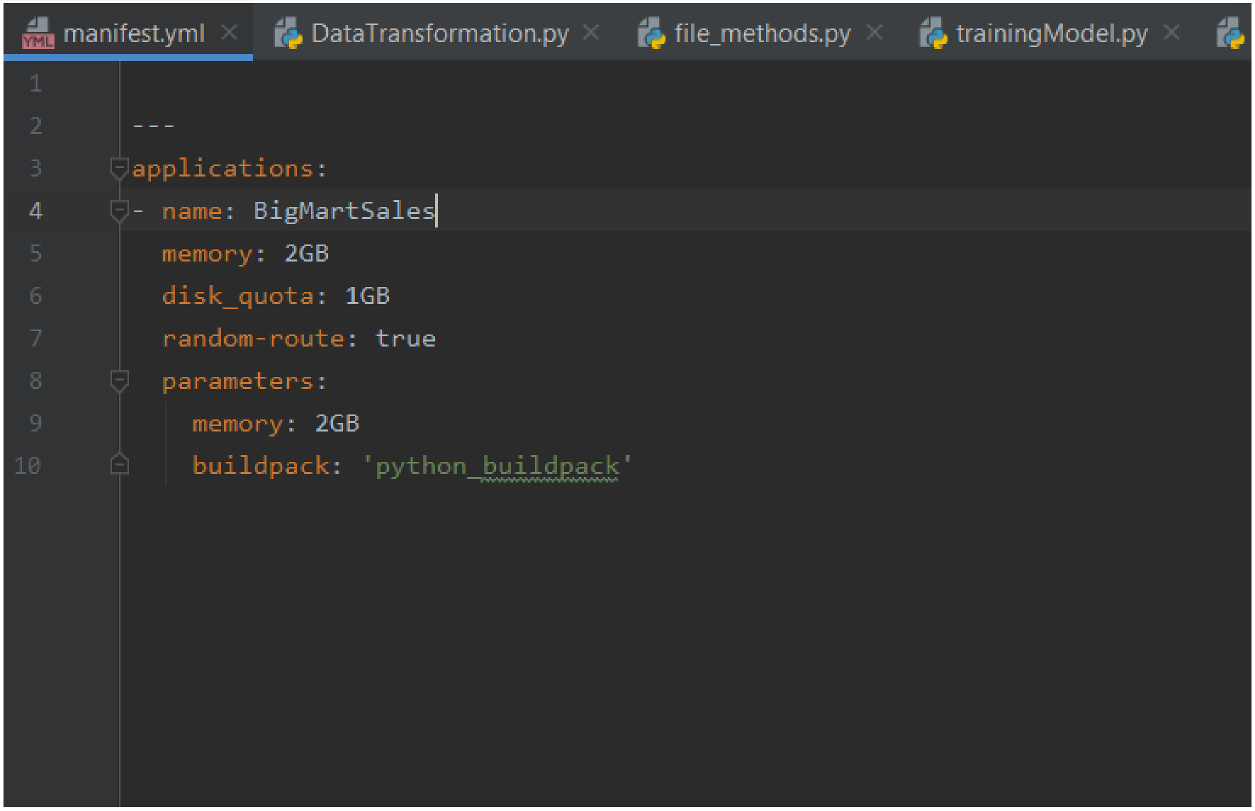
```
app.route("/predict", methods=['POST'])
cross_origin()
ef predictRouteClient():
    try:
        if request.json['folderPath'] is not None:
            path = request.json['folderPath']

            pred_val = pred_validation(path) #object initialization

            pred_val.prediction_validation() #calling the prediction_validation function

            pred = prediction(path) #object initialization

            # predicting for dataset present in database
            path = pred.predictionFromModel()
            return Response("Prediction File created at %s!!!" % path)

    except ValueError:
        return Response("Error Occurred! %s" %ValueError)
    except KeyError:
        return Response("Error Occurred! %s" %KeyError)
    except Exception as e:
        return Response("Error Occurred! %s" %e)
```

main.py is the entry point of our application, where the flask server starts.



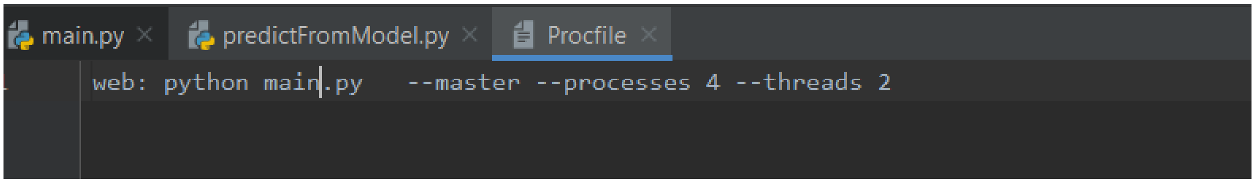
```python
from data_ingestion import data_loader_prediction
from application_logging import logger
from Prediction_Raw_Data_Validation.predictionDataValidation import Prediction_Data_validation



class prediction:

    def __init__(self,path):
        self.file_object = open("Prediction_Logs/Prediction_Log.txt", 'a+')
        self.log_writer = logger.App_Logger()
        self.pred_data_val = Prediction_Data_validation(path)

    def predictionFromModel(self):

        try:
            self.pred_data_val.deletePredictionFile() #deletes the existing prediction file from last run!
            self.log_writer.log(self.file_object,'Start of Prediction')
            data_getter=data_loader_prediction.Data_Getter_Pred(self.file_object,self.log_writer)
            data=data_getter.get_data()

            #code change
            # wafer_names=data['Wafer']
            # data=data.drop(labels=['Wafer'],axis=1)

            preprocessor=preprocessing.Preprocessor(self.file_object,self.log_writer)

            is_null_present,cols_with_missing_values=preprocessor.is_null_present(data)
            if(is_null_present):
                data=preprocessor.impute_missing_values(data)

            #data  = preprocessor.logTransformation(data)
            cols_to_drop = ["Item_Identifier", "Outlet_Identifier"]
            data_useful = preprocessor.remove_columns(data, cols_to_drop)
```

This is the **predictionFromModel.py** file where the predictions take place based on the data we are giving input to the model.



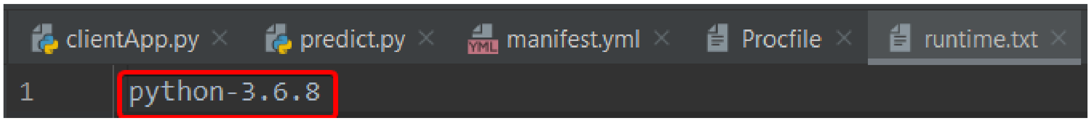
```yaml

---
applications:
- name: BigMartSales
  memory: 2GB
  disk_quota: 1GB
  random-route: true
  parameters:
    memory: 2GB
    buildpack: 'python_buildpack'
```

manifest.yml:- This file contains the instance configuration, app name, and build pack language.



```
web: python main.py   --master --processes 4 --threads 2
```

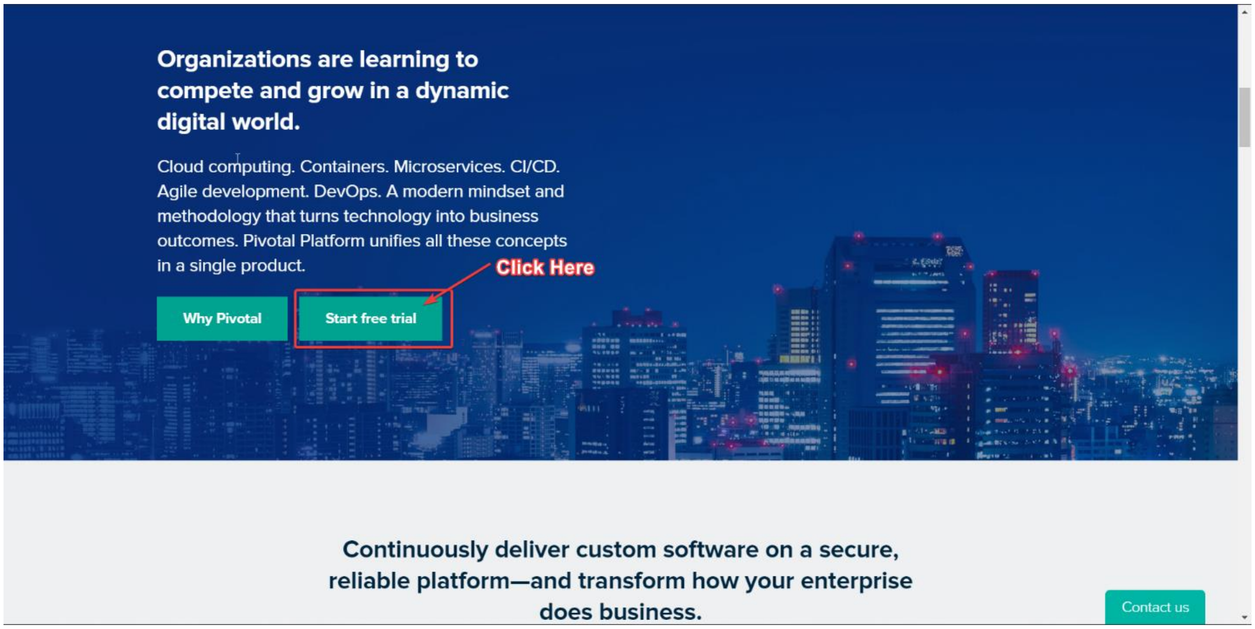
Procfile :- It contains the entry point of the app.



```
python-3.6.8
```

runtime.txt:- It contains the Python version number.

Visit the official website **https://pivotal.io/platform.**

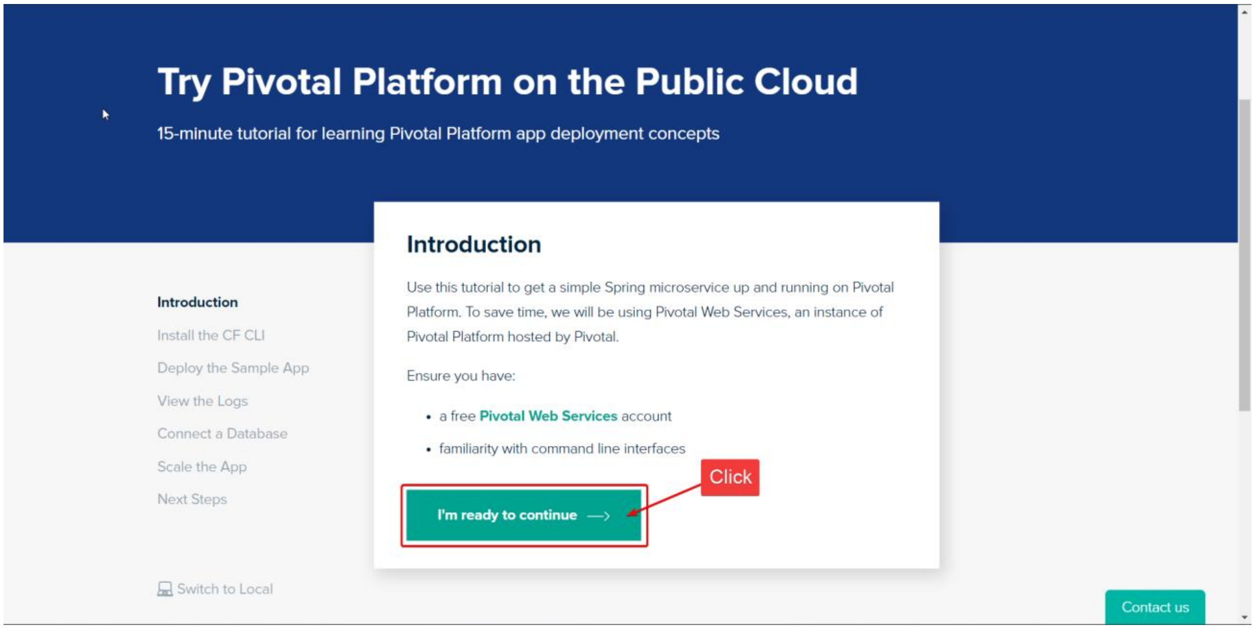


Organizations are learning to compete and grow in a dynamic digital world.

Cloud computing. Containers. Microservices. CI/CD. Agile development. DevOps. A modern mindset and methodology that turns technology into business outcomes. Pivotal Platform unifies all these concepts in a single product.

Continuously deliver custom software on a secure, reliable platform—and transform how your enterprise does business.

Scroll Down to see the **Start Trial Button**



Try Pivotal Platform on the Public Cloud

15-minute tutorial for learning Pivotal Platform app deployment concepts

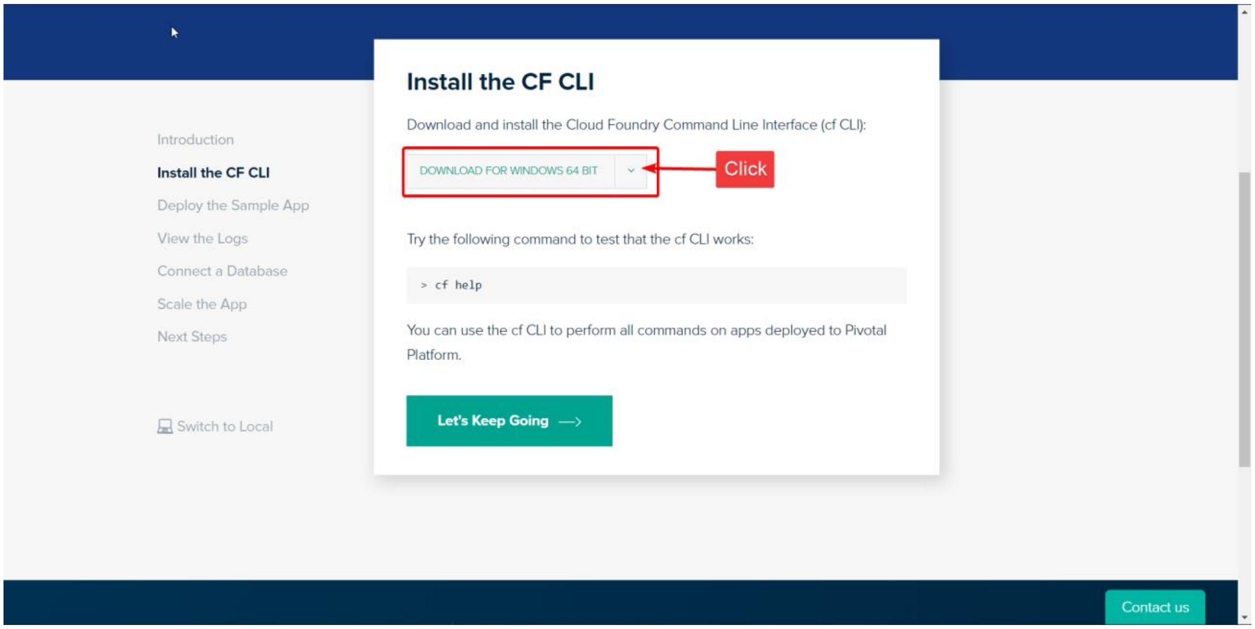
Introduction

Use this tutorial to get a simple Spring microservice up and running on Pivotal Platform. To save time, we will be using Pivotal Web Services, an instance of Pivotal Platform hosted by Pivotal.

Ensure you have:

- a free Pivotal Web Services account
- familiarity with command line interfaces

Click on the start trial button and the next interface will open. Then I will click on **I'm ready to continue**



Install the CF CLI

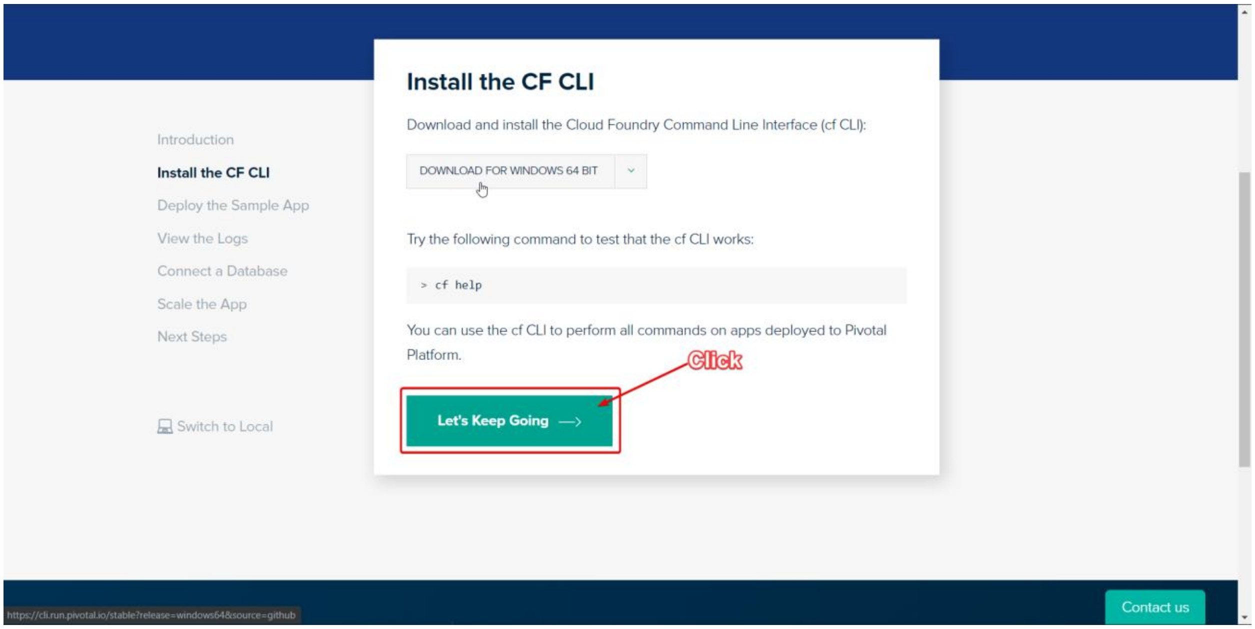
Download and install the Cloud Foundry Command Line Interface (cf CLI):

Try the following command to test that the cf CLI works:

```
> cf help
```

You can use the cf CLI to perform all commands on apps deployed to Pivotal Platform.

Click on Download for **Windows 64 bit, and** then zip file will be downloaded. Keep it for future uses.



Install the CF CLI

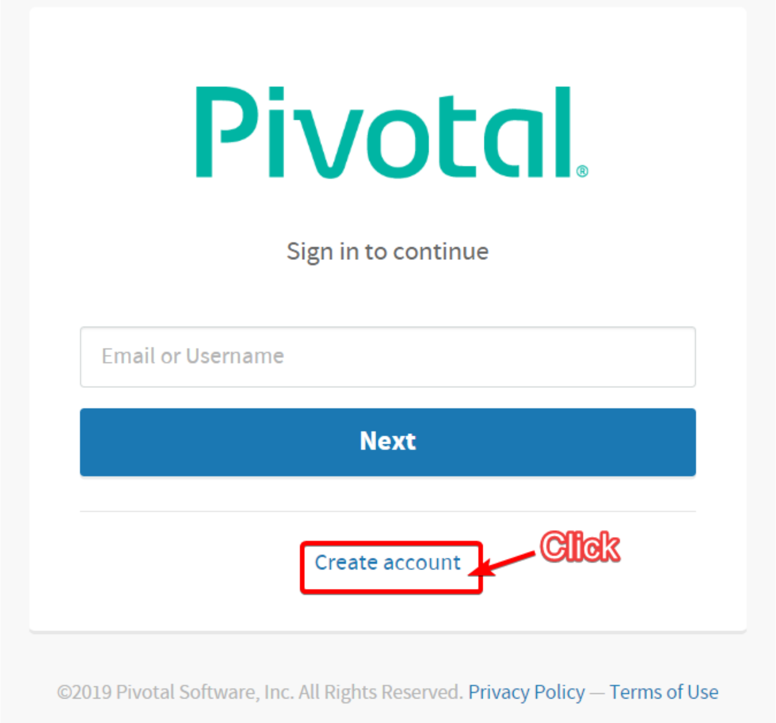
Download and install the Cloud Foundry Command Line Interface (cf CLI):

Try the following command to test that the cf CLI works:

```
> cf help
```

You can use the cf CLI to perform all commands on apps deployed to Pivotal Platform.

Now click on **Let’s Keep Going**

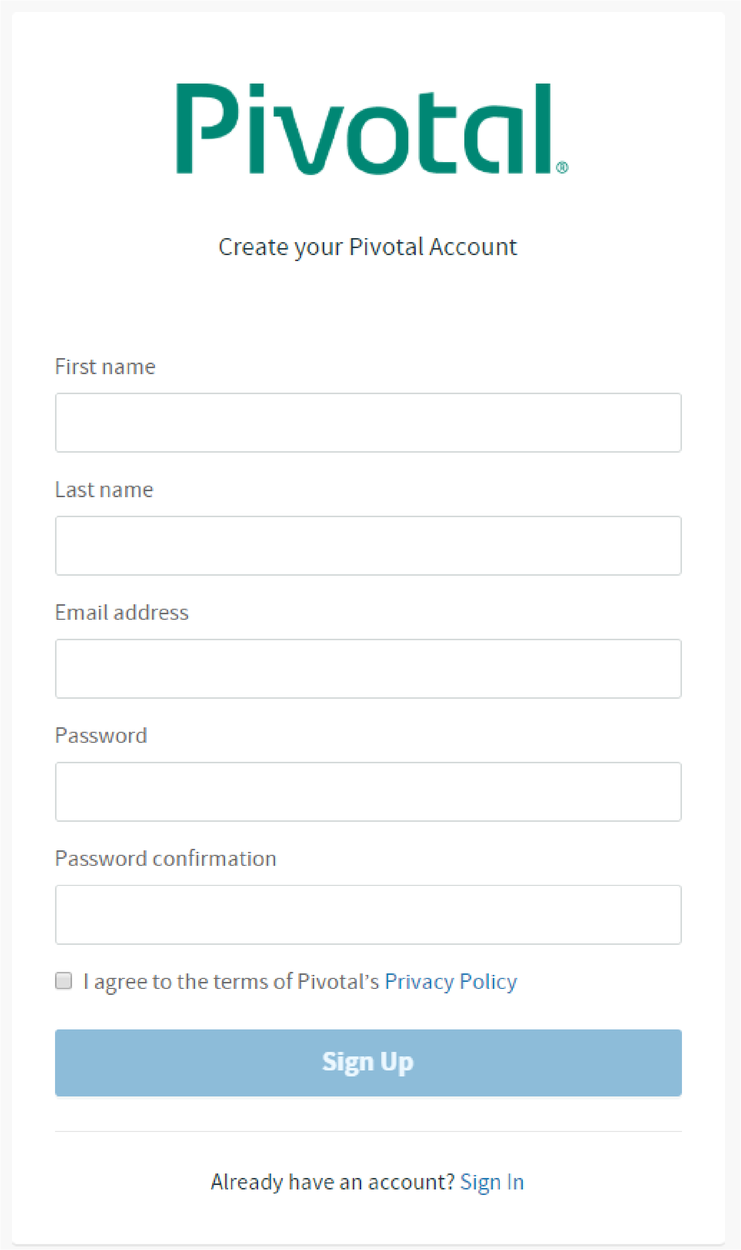


Sign in to continue

Create account

©2019 Pivotal Software, Inc. All Rights Reserved. Privacy Policy — Terms of Use

Click on **Create Your Account**



Pivotal

Create your Pivotal Account

First name

Last name

Email address

Password

Password confirmation

□ I agree to the terms of Pivotal's Privacy Policy

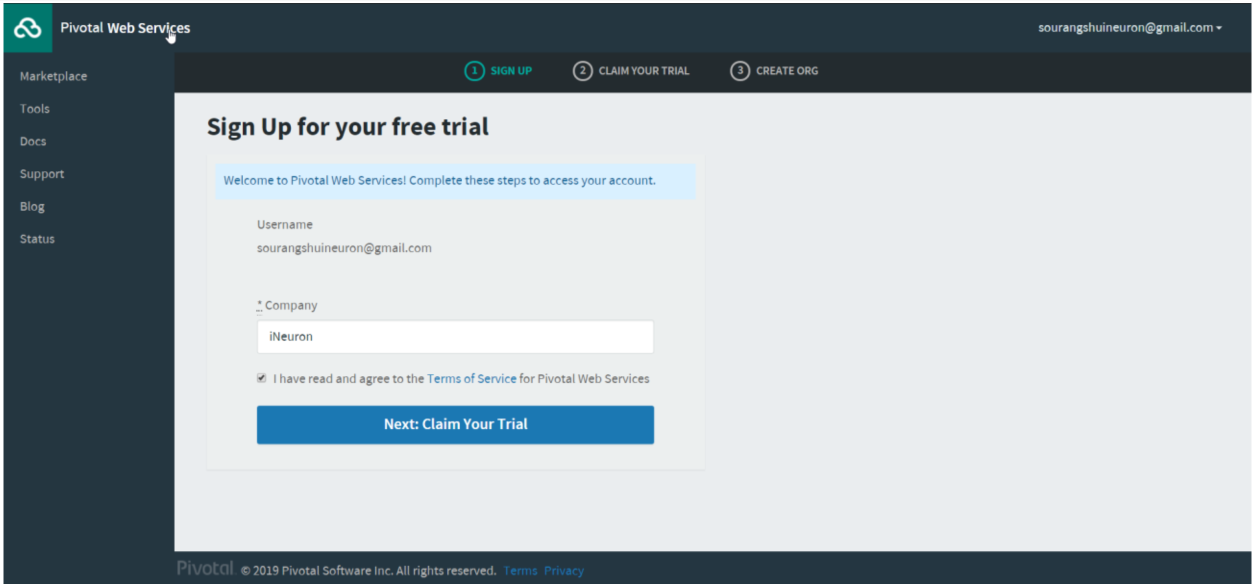
Sign Up

Already have an account? Sign In

Fill Up your Details For registration

Do the email verification

Then login in again



Pivotal Web Services

sourangshuineuron@gmail.com ▾

Sign Up for your free trial

Welcome to Pivotal Web Services! Complete these steps to access your account.

Username

sourangshuineuron@gmail.com

* Company

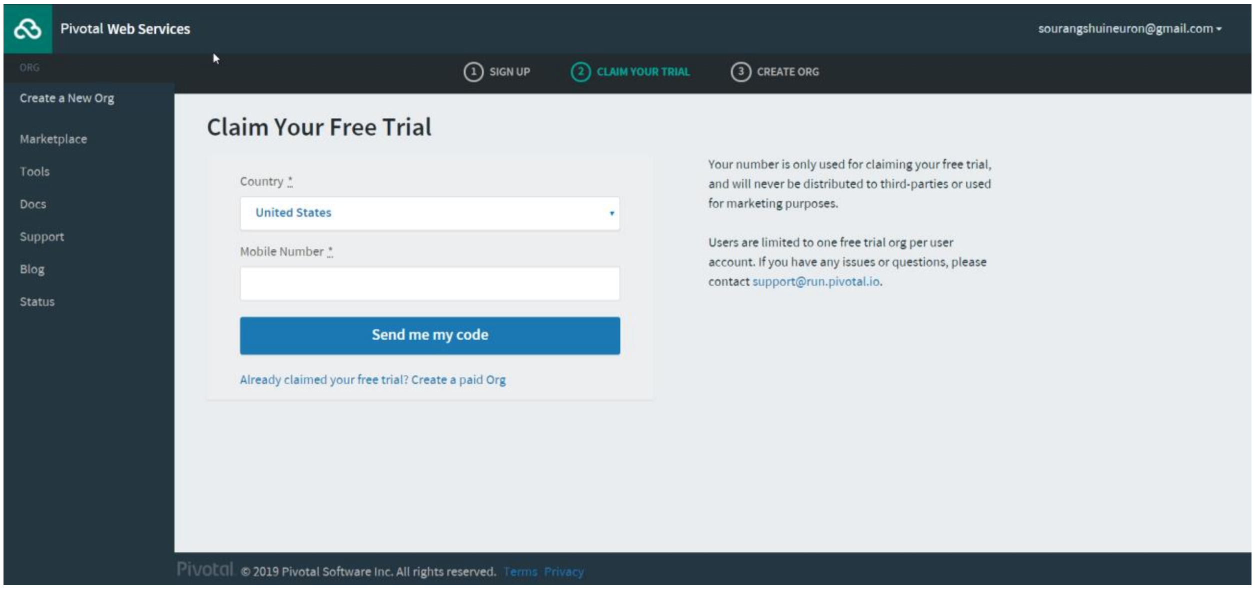
iNeuron

☑ I have read and agree to the Terms of Service for Pivotal Web Services

Pivotal © 2019 Pivotal Software Inc. All rights reserved. Terms Privacy

After logging you will see this screen below and start your free trial.

Write any Company or which one you prefer



Pivotal Web Services

sourangshuineuron@gmail.com ▾

Claim Your Free Trial

Country *

United States ▾

Mobile Number *

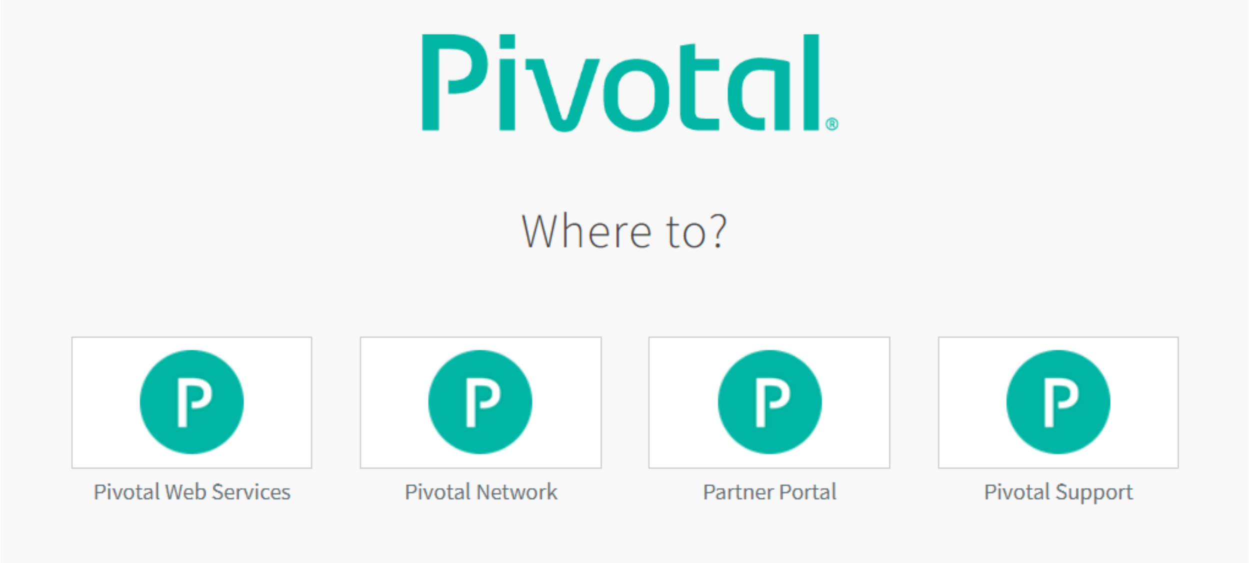
Already claimed your free trial? Create a paid Org

Your number is only used for claiming your free trial, and will never be distributed to third-parties or used for marketing purposes.

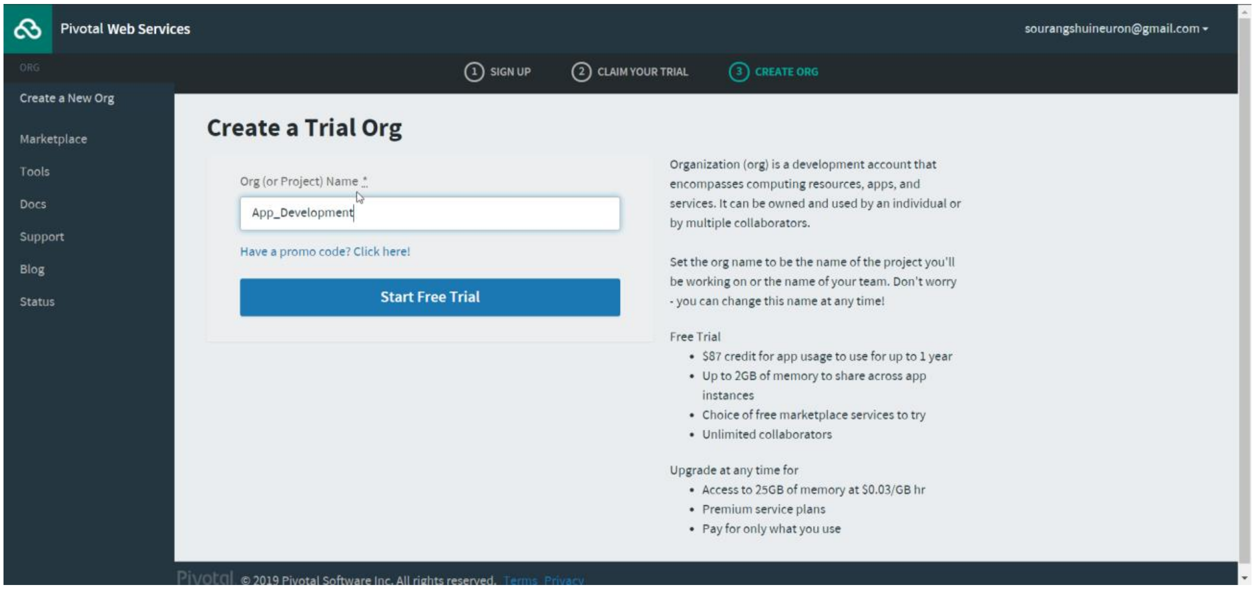
Users are limited to one free trial org per user account. If you have any issues or questions, please contact support@run.pivotal.io.

Pivotal © 2019 Pivotal Software Inc. All rights reserved. Terms Privacy

Enter your **Mobile Number** for Verification



Click on **Pivotal Web Services**



Create a Trial Org

Org (or Project) Name *: App_Development

Organization (org) is a development account that encompasses computing resources, apps, and services. It can be owned and used by an individual or by multiple collaborators.

Set the org name to be the name of the project you'll be working on or the name of your team. Don't worry - you can change this name at any time!

Free Trial

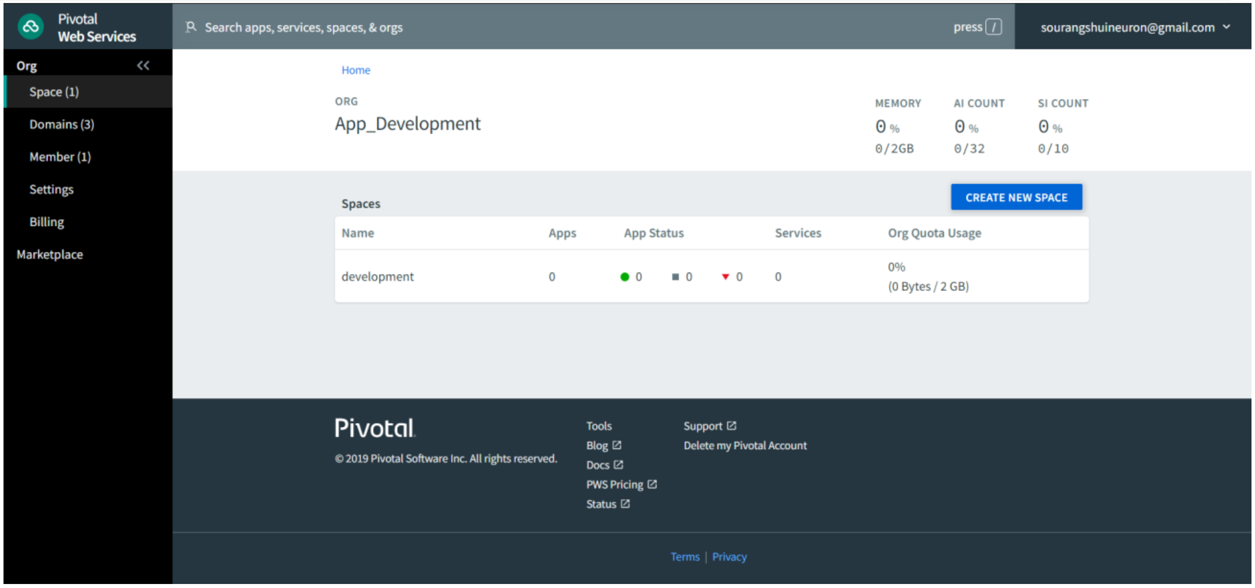
- $87 credit for app usage to use for up to 1 year
- Up to 2GB of memory to share across app instances
- Choice of free marketplace services to try
- Unlimited collaborators

Upgrade at any time for

- Access to 25GB of memory at $0.03/GB hr
- Premium service plans
- Pay for only what you use

© 2019 Pivotal Software Inc. All rights reserved.

Give any **Org** name



ORG
App_Development

MEMORY	AI COUNT	SI COUNT
0 % 0/2GB	0 % 0/32	0 % 0/10

Spaces

Name	Apps	App Status	Services	Org Quota Usage
development	0	● 0 ■ 0 ▼ 0	0	0% (0 Bytes / 2 GB)

© 2019 Pivotal Software Inc. All rights reserved.

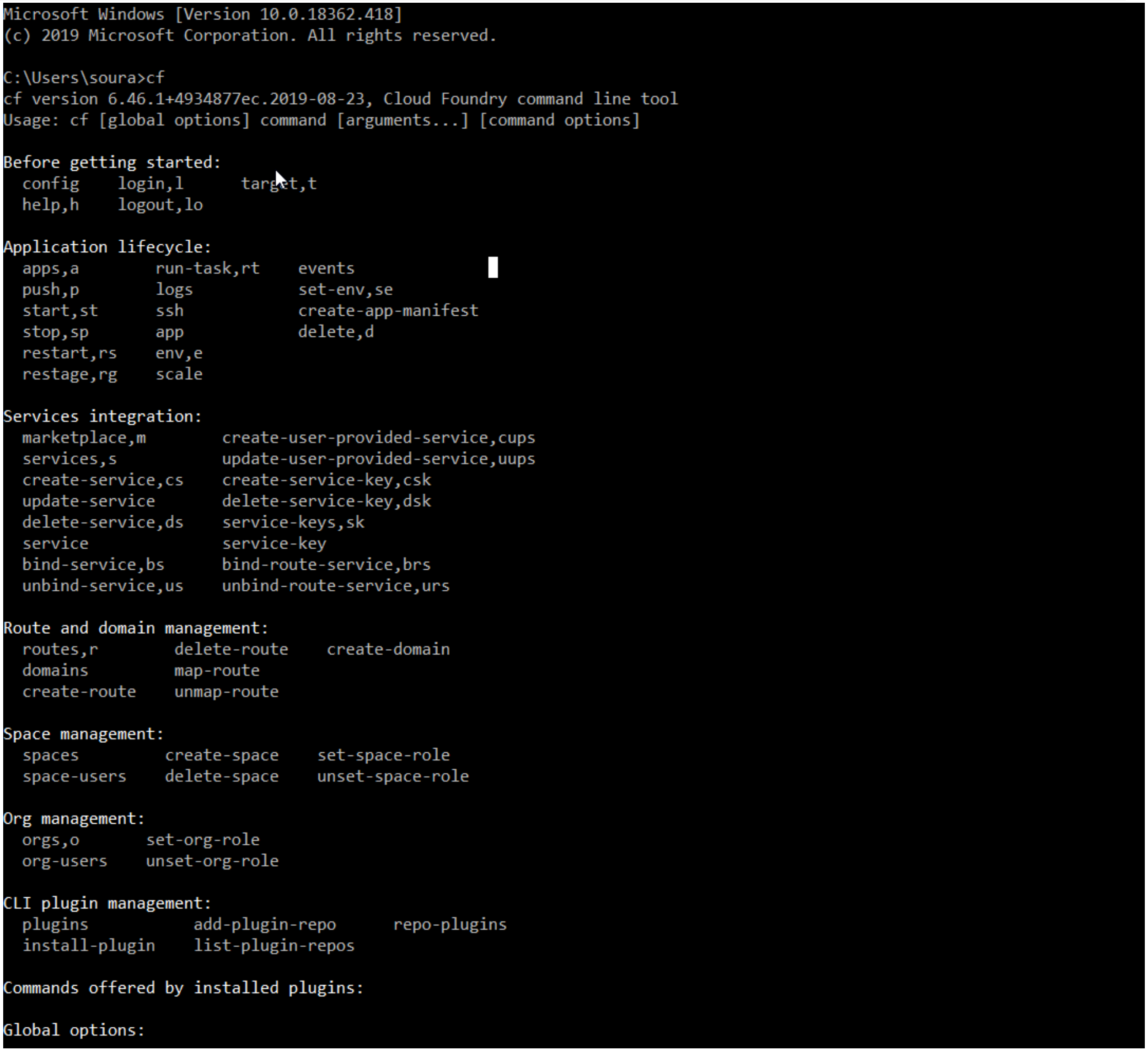
Now you are inside your Org, and by default, development space is created in your org. You can push your apps here.

The cloud signup process is done, and the setup is ready for us to push the app.

Previously you have downloaded the **CLI.zip** file. Unzip the file and install the .exe file with admin rights.

After a successful installation, you can verify by opening your CMD and type **cf**.

Then you will get a screen which is shown below



```
Microsoft Windows [Version 10.0.18362.418]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\soura>cf
cf version 6.46.1+4934877ec.2019-08-23, Cloud Foundry command line tool
Usage: cf [global options] command [arguments...] [command options]

Before getting started:
  config    login,l      target,t
  help,h    logout,lo

Application lifecycle:
  apps,a        run-task,rt    events
  push,p        logs           set-env,se
  start,st      ssh            create-app-manifest
  stop,sp       app            delete,d
  restart,rs    env,e
  restage,rg    scale

Services integration:
  marketplace,m        create-user-provided-service,cups
  services,s           update-user-provided-service,uups
  create-service,cs    create-service-key,csk
  update-service       delete-service-key,dsk
  delete-service,ds    service-keys,sk
  service              service-key
  bind-service,bs      bind-route-service,brs
  unbind-service,us    unbind-route-service,urs

Route and domain management:
  routes,r        delete-route    create-domain
  domains         map-route
  create-route    unmap-route

Space management:
  spaces         create-space    set-space-role
  space-users    delete-space    unset-space-role

Org management:
  orgs,o       set-org-role
  org-users    unset-org-role

CLI plugin management:
  plugins           add-plugin-repo      repo-plugins
  install-plugin    list-plugin-repos

Commands offered by installed plugins:

Global options:
```

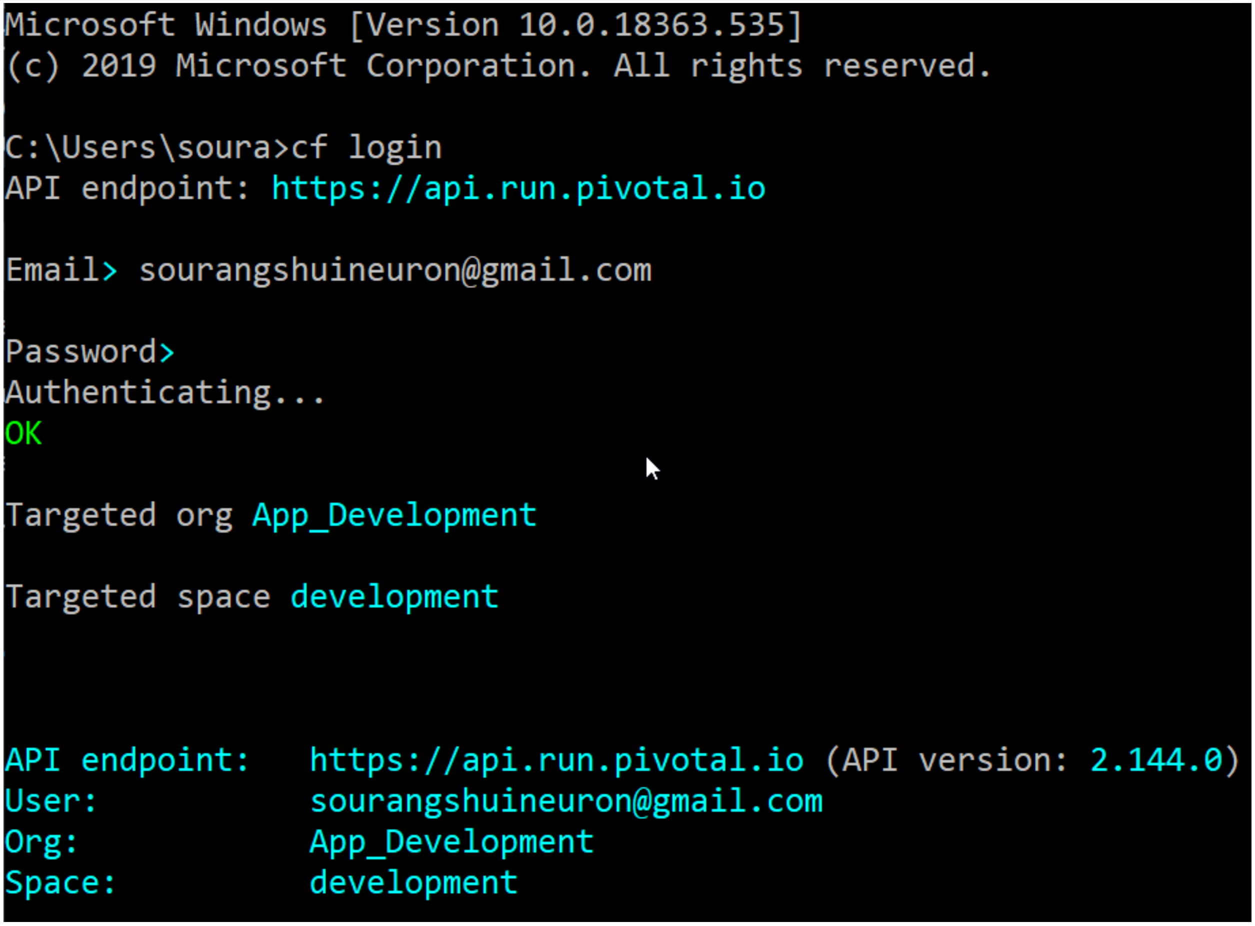
If you see this screen in your CMD, the installation is successful.

Now type the command to login via cf-cli

cf login -a https://api.run.pivotal.io

Next, enter your email and password. Now you are ready to push your app.

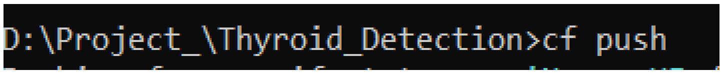
Now let’s go to the app which we have built.



```
Microsoft Windows [Version 10.0.18363.535]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\soura>cf login
API endpoint: https://api.run.pivotal.io

Email> sourangshuineuron@gmail.com

Password>
Authenticating...
OK

Targeted org App_Development

Targeted space development



API endpoint:   https://api.run.pivotal.io (API version: 2.144.0)
User:           sourangshuineuron@gmail.com
Org:            App_Development
Space:          development
```

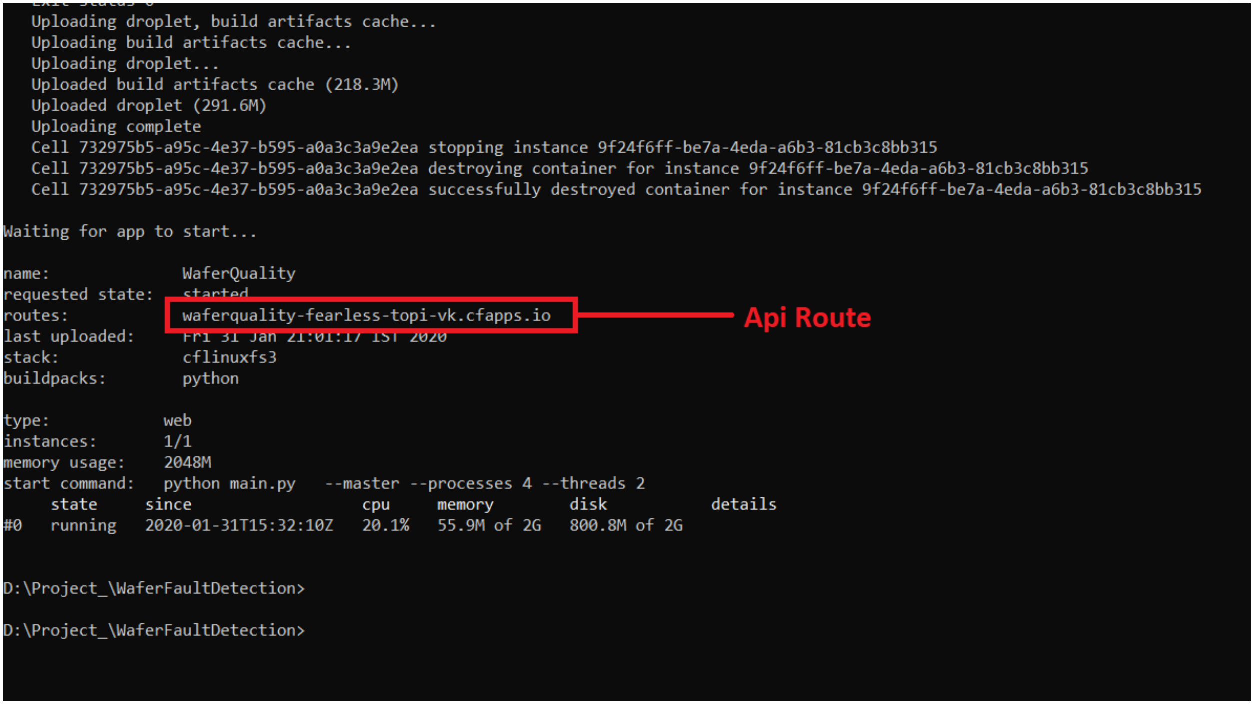
Navigate to the project folder after downloading from the given below link:-

Then write cf push in the terminal.



```
D:\Project_\Thyroid_Detection>cf push
```

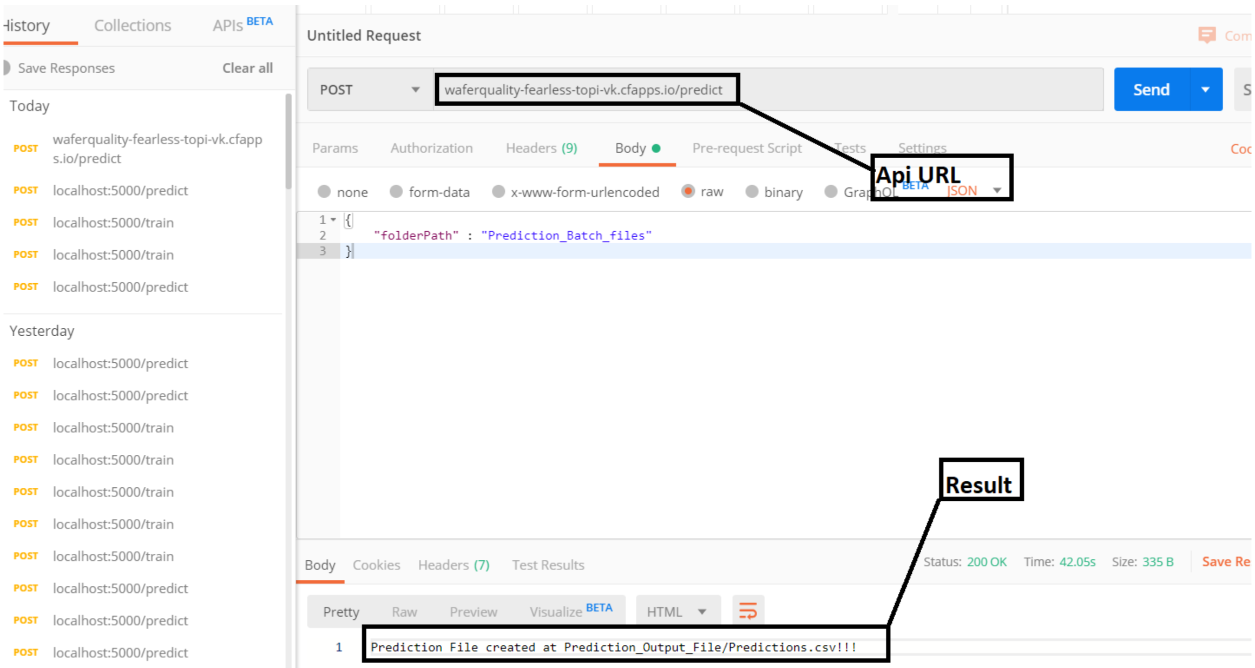
After the app is successfully deployed in the cloud, you will see the screen below with the route.



```
   Uploading droplet, build artifacts cache...
   Uploading build artifacts cache...
   Uploading droplet...
   Uploaded build artifacts cache (218.3M)
   Uploaded droplet (291.6M)
   Uploading complete
   Cell 732975b5-a95c-4e37-b595-a0a3c3a9e2ea stopping instance 9f24f6ff-be7a-4eda-a6b3-81cb3c8bb315
   Cell 732975b5-a95c-4e37-b595-a0a3c3a9e2ea destroying container for instance 9f24f6ff-be7a-4eda-a6b3-81cb3c8bb315
   Cell 732975b5-a95c-4e37-b595-a0a3c3a9e2ea successfully destroyed container for instance 9f24f6ff-be7a-4eda-a6b3-81cb3c8bb315

Waiting for app to start...

name:              WaferQuality
requested state:   started
routes:            waferquality-fearless-topi-vk.cfapps.io
last uploaded:     Fri 31 Jan 21:01:17 IST 2020
stack:             cflinuxfs3
buildpacks:        python

type:           web
instances:      1/1
memory usage:   2048M
start command:  python main.py   --master --processes 4 --threads 2
     state     since                  cpu     memory         disk           details
#0   running   2020-01-31T15:32:10Z   20.1%   55.9M of 2G    800.8M of 2G


D:\Project_\WaferFaultDetection>

D:\Project_\WaferFaultDetection>
```

Finally, the app is pushed in the cloud.

Lets Open Postman and see the result.



Today

POST waferquality-fearless-topi-vk.cfapps.io/predict
POST localhost:5000/predict
POST localhost:5000/train
POST localhost:5000/train
POST localhost:5000/predict

Yesterday

POST localhost:5000/predict
POST localhost:5000/predict
POST localhost:5000/train
POST localhost:5000/train
POST localhost:5000/train
POST localhost:5000/train
POST localhost:5000/train
POST localhost:5000/predict
POST localhost:5000/predict
POST localhost:5000/predict

POST waferquality-fearless-topi-vk.cfapps.io/predict

```
{
    "folderPath" : "Prediction_Batch_files"
}
```

Status: 200 OK   Time: 42.05s   Size: 335 B

```
Prediction File created at Prediction_Output_File/Predictions.csv!!!
```